Lenguajes Formales y Computabilidad
Resumen Teorico
Parcial 3
Guias 5 a 7

Agustín M. Domínguez

1C 2025

Índice

Este apunte tiene las definiciones importantes de la materia, sin varias de las explicaciones o demostraciones que están en las guías, a modo de *resumen*.

Esta materia tiene la particularidad que (aparte de las clases mismas) las guías son el recurso más completo para el estudiante, que funcionan como **Resumen de la clase**, **Apunte Teórico**, y **Práctico de la materia** en un solo documento. Otra particularidad es que las guías son documentos vivos, en el sentido que los profes están constantemente expandiendo y ajustando el contenido, por lo que este documento *puede* estar desactualizado si se consulta en el futuro. Siempre conviene buscar la última versión de las guías y estudiar desde ahí. Sin embargo, este documento puede servir para tener las definiciones importantes de la materia a mano.

Guía 5: El Paradigma de Gödel (Parte 2)

1.1 Cuantificación Acotada de Predicados

La cuantificación acotada es una idea de probar propiedades (es decir, un predicado) que se cumplen para todo elementos ($\forall$) del espacio, o que existe un elemento del espacio la cual se cumple ($\exists$). La idea de "acotar" surge de que es imposible probar que cierta propiedad se cumple para absolutamente todo el espacio, pero sí es posible probar lo mismo si limitamos, o acotamos, el espacio.

Vamos a tener los siguientes casos a estudiar:

- Probar una propiedad sobre una variable cuantificada **numérica** que se cumple para **todo elemento** del espacio ($\forall$)
- Probar una propiedad sobre una variable cuantificada **numérica** que se cumple para **al menos un elemento** del espacio ($\exists$)
- Probar una propiedad sobre una variable cuantificada **alfabética** que se cumple para **todo elemento** del espacio ($\forall$)
- Probar una propiedad sobre una variable cuantificada **alfabética** que se cumple para **al menos un elemento** del espacio ($\exists$)

Def: $\overline{S}$ **y** $\overline{L}$

Sea

$$P \ : \ S \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$$

un predicado, con $S, S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \Sigma^*$
y sea

$$P \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times L \to \omega$$

un predicado, con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m, L \subseteq \Sigma^*$
Llamaremos $\overline{S}$ a un conjunto $\overline{S} \subseteq S$ y $\overline{L}$ a un conjunto $\overline{L} \subseteq L$

Def: **Cuantificacion acodatada de predicados Σ-PR con dominio rectangular**

La expresión booleana

$$(\forall t \in \overline{S})_{t \leq x}\ P(t, \overrightarrow{x}, \overrightarrow{\alpha})$$

depende de las variables $x, \overrightarrow{x}, \overrightarrow{\alpha}$ y valdrá 1 en una (1+n+m)-upla $(x, \overrightarrow{x}, \overrightarrow{\alpha})$ cuando $P(t, \overrightarrow{x}, \overrightarrow{\alpha})$ sea igual a 1 para cada $t \in \{u \in S \mid u \leq x\}$; y 0 en el caso contrario.
Tenemos entonces el que el dominio del predicado:

$$\lambda x\, \overrightarrow{x}\, \overrightarrow{\alpha}[(\forall t \in \overline{S})_{t \leq x}\ P(t, \overrightarrow{x}, \overrightarrow{\alpha})]$$

es $\omega \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$.
En forma análoga se define la forma de interpretar la expresion booleana

$$(\exists t \in \overline{S})_{t \leq x}\ P(t, \overrightarrow{x}, \overrightarrow{\alpha})$$

Lemma: **Cuantificacion acotada (Parte 1)**

Sea Σ un alfabeto finito y sea

$$P\ :\ S \times S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \to \omega$$

un predicado Σ-PR con $S, S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m \subseteq \Sigma^*$ no vacíos.
Supongamos $\overline{S} \subseteq S$ es Σ-PR
Entonces:

$$\lambda x\, \overrightarrow{x}\, \overrightarrow{\alpha}[(\forall t \in \overline{S})_{t \leq x}\ P(t, \overrightarrow{x}, \overrightarrow{\alpha})]$$

y

$$\lambda x\, \overrightarrow{x}\, \overrightarrow{\alpha}[(\exists t \in \overline{S})_{t \leq x}\ P(t, \overrightarrow{x}, \overrightarrow{\alpha})]$$

son predicados Σ-PR

Lemma: **Cuantificacion acotada (Parte 2)**

Sea Σ un alfabeto finito y sea

$$P \ : \ S_1 \times \cdots \times S_n \times L_1 \times \cdots \times L_m \times L \to \omega$$

un predicado Σ-PR, con $S_1, \ldots, S_n \subseteq \omega$ y $L_1, \ldots, L_m, L \subseteq \Sigma^*$ no vacíos
Supongamos $\overline{L} \subseteq L$ es Σ-PR
Entonces:

$$\lambda x \, \overrightarrow{x} \, \overrightarrow{\alpha} [(\forall \alpha \in \overline{L})_{|\alpha| \leq x} \ P(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha)]$$

y

$$\lambda x \, \overrightarrow{x} \, \overrightarrow{\alpha} [(\exists \alpha \in \overline{L})_{|\alpha| \leq x} \ P(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha)]$$

son predicados Σ-PR

Notar que en este caso acotamos sobre el *largo* de la palabra.

La acotación es necesaria porque existe un predicado P que es Σ-PR pero que:

$\lambda \alpha [(\exists t \in \omega) P(t, \alpha)]$

NO es Σ efectivamente computable, y por lo tanto tampoco es Σ-recursivo ni menos que menos Σ-PR. Este predicado en particular se ve más adelante en la materia.

La idea fundamental en la aplicacion de las propiedades que nos interesan es que en muchos casos de predicados obtenidos por cuantificacion a partir de otros predicados, la variable cuantificada tiene una cota natural en terminos de las otras variables y entonces componiendo adecuadamente se lo puede presentar como un caso de cuantificacion acotada

Guía 6: El Paradigma de Gödel (Parte 3)

2.1 Minimización y Funciones Σ-recursivas

En esta parte vemos el último constructor de Gödel para crear todo el mundo de funciones computables: El constructor *minimización*.

Tiene dos casos aunque solo usaremos el primero para la definicion de funcion Σ-recursiva

Def: **Expresión** $\min_t$
Sea Σ un alfabeto finito y sea
P tal que $D_P \subseteq \omega \times \omega^n \times \Sigma^{*m} \to \omega$ un predicado.
Dado $(\overrightarrow{x}, \overrightarrow{\alpha}) \in \omega^n \times \Sigma^{*m}$, cuando exista al menos un $t \in \omega$ tal que $P(t, \overrightarrow{x}, \overrightarrow{\alpha}) = 1$, usaremos $\min_t$ para denotar al menor de tales t's
Esta expresión está definida **solo** para aquellas $(n+m)$-uplas $(\overrightarrow{x}, \overrightarrow{\alpha})$ para las cuales existe un menor t que cumple la propiedad.
En otras palabras. $\min_t P(t, \overrightarrow{x}, \overrightarrow{\alpha})$ es una expresión que **no** está definida si $(t, \overrightarrow{x}, \overrightarrow{\alpha})$ no pertenece a D_P o si $P(t, \overrightarrow{x}, \overrightarrow{\alpha}) = 0 \ \forall t \in \omega$

Def: **Minimización**
Dado un predicado P
definimos la minimización de P como:

$$M(P) = \lambda[\min_t \ P(t, \overrightarrow{x}, \overrightarrow{\alpha})]$$

Notar que el dominio de $M(P)$ es:

$$D_{M(P)} = \{(\overrightarrow{x}, \overrightarrow{\alpha}) \in \omega^n \times \Sigma^{*m} \mid (\exists t \in \omega) \ P(t, \overrightarrow{x}, \overrightarrow{\alpha})\}$$

Esta es *minimización de variable numérica a partir de P*. El otro caso sería sobre variable alfabética, el cual vemos más adelante.

Def: **Regla U**

Si tiene una función $f : D_f \subseteq \omega^n \times {\Sigma^*}^m \to \omega$ y busca un predicado P tal que $f = M(P)$, intentar diseñar P de manera que para cada $(\vec{x}, \vec{\alpha}) \in D_f$ se de que:

$$f(\vec{x}, \vec{\alpha}) = \text{unico } t \in \omega \text{ tal que } P(t, \vec{x}, \vec{\alpha})$$

Lemma:

Si $P : D_P \subseteq \omega \times \omega^n \times {\Sigma^*}^m \to \omega$ es un predicado Σ-efectivamente computable y D_P es Σ-efectivamente computable, entonces la función $M(P)$ es Σ-efectivamente computable

Corolario:

Si $P : \omega \times \omega^n \times {\Sigma^*}^m \to \omega$ es un predicado Σ-efectivamente computable, entonces la función $M(P)$ es Σ-efectivamente computable

Def: **Función Σ recursiva**

Se define de manera similar a Σ-PR
$R_0^\Sigma = \mathrm{PR}_0^\Sigma$
$A = \{f \circ [f_1, \ldots, f_n] : f, f_1, \ldots, f_r \in R_k^\Sigma, r \geq 1\}$
$B = \{R(f, \mathcal{G}) : R(f, \mathcal{G}) \text{ esta definida y } \{f\} \cup \{\mathcal{G}_a : a \in \Sigma\} \subseteq R_k^\Sigma\}$
$C = \{R(f, g) : R(f, g) \text{ esta definida y } f, g \in P_k^\Sigma\}$
$D = \{M(P) : P \text{ es un predicado } \Sigma\text{-total y } P \in R_k^\Sigma\}$
$R_{k+1}^\Sigma = \mathrm{PR}_0^\Sigma \bigcup A \bigcup B \bigcup C \bigcup D$
$R^\Sigma = \bigcup_{k \geq 0} R_k^\Sigma$
una función f e llama Σ-recursiva si pertenece a R^Σ

Proposición: **Leibniz vence a Gödel**

$F \in R^\Sigma \implies F$ es Σ-efectivamente computable

Proposición:

Sea Σ un alfabeto finito. Entonces no toda función Σ-recursiva es Σ-PR.
Dicho de otra forma:

$$\mathrm{PR}^\Sigma \neq R^\Sigma$$

Lemma: **Minimización acotada**

Sean $n, m \geq 0$. Sea $P : D_P \subseteq \omega \times \omega^n \times {\Sigma^*}^m \to \omega$ un predicado Σ-PR. Entonces:

- **(a)** $M(P)$ es Σ-recursiva
- **(b)** Si hay una función Σ-PR $f : \omega^n \times {\Sigma^*}^m \to \omega$ tal que
  $M(P)(\vec{x}, \vec{\alpha}) = \min_t P(t, \vec{x}, \vec{\alpha}) \leq f(\vec{x}, \vec{\alpha}) \quad \forall (\vec{x}, \vec{\alpha}) \in D_{M(P)}$

entonces $M(P)$ es Σ-PR

La idea del lemma anterior es que las funciones Σ-R que **no** son Σ-PR comparten la característica de que *'crecen demasiado rápido'*. Entonces si podemos acotar el crecimiento de una

minimización con una función que sí es Σ-PR, entonces sabemos que la minimización es también Σ-PR

Algunas funciones que son Σ-PR

A continuación listamos algunas funciones que en la guía se prueban que son Σ-PR a partir de los resultados anteriores.

$$\begin{aligned} Q &: \omega \times \mathbb{N} \to \mathbb{N} \\ &(x, y) \to \text{cociente de la división de x por y} \end{aligned} \tag{2.1}$$

$$\begin{aligned} R &: \omega \times \mathbb{N} \to \mathbb{N} \\ &(x, y) \to \text{resto de la división de x por y} \end{aligned} \tag{2.2}$$

$$\begin{aligned} pr &: \mathbb{N} \to \omega \\ &n \to n\text{-esimo numero primo} \end{aligned} \tag{2.3}$$

Def: **Minimización de variable alfabética**

Supongamos que $\Sigma \neq \varnothing$. Sea $\leq$ un orden total sobre Σ.
Sea $P : D_P \subseteq \omega^n \times \Sigma^{*m} \times \Sigma^* \to \omega$ un predicado.
Cuando $(\vec{x}, \vec{\alpha}) \in \omega^n \times \Sigma^{*m}$ es tal que existe al menos un $\alpha \in \Sigma^*$ tal que $P(\vec{x}, \vec{\alpha}, \alpha) = 1$ usaremos:

$$\min_{\alpha}^{\leq} P(\vec{x}, \vec{\alpha}, \alpha)$$

para denotar al menor $\alpha \in \Sigma^*$ tal que $P(\vec{x}, \vec{\alpha}, \alpha) = 1$
Es decir que dicha expresión **solo está definida cuando existe al menos un elemento que cumpla la propiedad**.
Entonces definimos:

$$M^{\leq}(P) = \lambda \vec{x}\, \vec{\alpha} \left[\min_{\alpha}^{\leq} P(\vec{x}, \vec{\alpha}, \alpha)\right]$$

Notar que basado en lo anterior se cumple que:

$D_{M^{\leq}(P)} = \{(\vec{x}, \vec{\alpha}) \in \omega^n \times \Sigma^{*m} : (\exists\ \alpha \in \Sigma^*)\ P(\vec{x}, \vec{\alpha}, \alpha)\}$

Diremos que $M^{\leq}(P)$ es obtenida por *minimización de variable alfabética a partir de P*

Lemma: **Minimización acotada alfabética**

Supongamos $\Sigma \neq$ y $\leq$ orden total sobre Σ. Sean $n, m \geq 0$ y sea $P : D_P \subseteq \omega^n \times \Sigma^{*m} \times \Sigma^* \to \omega$ un predicado Σ-PR entonces:

- **(a)** $M^{\leq}(P)$ es Σ-recursiva
- **(b)** Si existe una función Σ-PR $f : \omega^n \times \Sigma^{*m} \to \omega$ tal que:
  $|M^{\leq}(P)(\vec{x}, \vec{\alpha})| = |\min_{\alpha}^{\leq} P(\vec{x}, \vec{\alpha}, \alpha)| \leq f(\vec{x}, \vec{\alpha}) \quad \forall\ (\vec{x}, \vec{\alpha}) \in D_{M^{\leq}(P)}$

entonces $M^{\leq}(P)$ es Σ-PR

Def: **Conjuntos Σ-resurivamente enumerables**

Diremos que un conjunto $S \subseteq \omega^n \times {\Sigma^*}^m$ es Σ-recursivamente enumerable cuando sea vacío o haya una función $F : \omega \to \omega^n \times {\Sigma^*}^m$ tal que $I_F = S$ y $F_{(i)}$ sea Σ-R para cada $i \in \{1, \ldots, n+m\}$

Def: **Conjuntos Σ-recursivos**

Un conjunto $S \subseteq \omega^n \times {\Sigma^*}^m$ es Σ-recursivo cuando la función $\chi_S^{\omega^n \times {\Sigma^*}^m}$ sea Σ-recursiva

Teo: **Independencia del alfabeto**

Sean Σ y Γ alfabetos finitos cualesquiera. Se cumple que:
Si f es Σ-mixta y Γ-mixta, entonces:

$$f \in \mathrm{PR}^{\Sigma} \iff f \in \mathrm{PR}^{\Gamma}$$

Similarmente, si S es Σ-mixto y Γ-mixto, entonces:

$$S \text{ es } \Sigma\text{-recursivo} \iff S \text{ es } \Gamma\text{-recursivo}$$

Guía 7: El Paradigma Imperativo de Neumann

3.1 Funciones con números

Primero un repaso y definiciones de funciones que van a ser necesarias:

Def: **Num**

$$\text{Num} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Def: **Sig**

$$Sig : Num^* \to Num^*$$
$$Sig(\varepsilon) = 1$$
$$Sig(\alpha 0) = \alpha 1$$
$$\dots$$
$$Sig(\alpha 8) = \alpha 9$$
$$Sig(\alpha 9) = Sig(\alpha)0$$

Def: **Dec**

Esta es la representación en 'palabra' de un número dado:

$$Dec : \omega \to Num^*$$
$$Dec(0) = \epsilon$$
$$Dec(n+1) = Sig(Dec(n))$$

Por notación, denotaremos $\overline{n}$ en lugar de $Dec(n)$
Es decir que $\overline{n}$ con $n \in \omega$ es una **palabra** que representa el *número* n

3.2 Definición y Sintaxis del lenguaje $\mathcal{S}^{\Sigma}$

En esta parte de la materia se definirá un lenguaje de programación basado de un alfabeto Σ. Lo llamaremos $\mathcal{S}^{\Sigma}$

3.2.1 Alfabeto de $\mathcal{S}^{\Sigma}$

La **sintaxis** de $\mathcal{S}^{\Sigma}$ será dada utilizando solo símbolos del alfabeto $\Sigma \bigcup \Sigma_P$ donde:

Def: Σ_P

$$\Sigma_P = Num \cup \{\leftarrow, +, \dot{-}, ., \neq, \rightsquigarrow, \varepsilon, \text{N, K, L, I, F, G, O, T, B, E, S}\}$$

3.2.2 Variables de $\mathcal{S}^{\Sigma}$

En la sintaxis también definiremos distintas variables de la siguiente forma:

Def: **Variables Numéricas**
Estas variables tendrán la forma la forma N seguida por una palabra $\overline{k}$ con $k \in \mathbb{N}$
Por ejemplo 'N1', 'N2', etc.
Estas variables siempre representarán números, por lo que las llamamos *variables numéricas de* $\mathcal{S}^{\Sigma}$

Def: **Variables Alfabéticas**
Estas variables tendrán la forma la forma P seguida por una palabra $\overline{k}$ con $k \in \mathbb{N}$
Por ejemplo 'P1', 'P2', etc.
Estas variables siempre representarán palabras, por lo que las llamamos *variables alfabéticas de* $\mathcal{S}^{\Sigma}$

Def: **Labels**
Estas variables tendrán la forma la forma L seguida por una palabra $\overline{k}$ con $k \in \mathbb{N}$
Por ejemplo 'L1', 'L2', etc.
Estas variables se utilizan para definir puntos relevantes del programa donde se pueden hacer **saltos de linea o control** ('GO TO')
A estas variables las llamamos Labels.

3.2.3 Intrucciones

Hay solo dos tipos de *intrucciones básicas* en nuestro lenguaje, una intrucción de control (GOTO) y una instrucción de asignación.

Def: **Asignación**

La asignación tiene la forma:

$$\text{VARIABLE} \leftarrow \text{OPERACION}$$

donde 'VARIABLE' es una variable numérica o alfabética y 'OPERACIÓN' es una de las opreaciones aceptadas.
A continuación se listan **todas** las asignaciones posibles:

- $N\overline{k} \leftarrow N\overline{k} + 1$
- $N\overline{k} \leftarrow N\overline{k} \dot{-} 1$
- $N\overline{k} \leftarrow N\overline{n}$
- $N\overline{k} \leftarrow 0$
- $P\overline{k} \leftarrow P\overline{k}.a \quad (\text{con } a \in \Sigma)$
- $P\overline{k} \leftarrow \rightsquigarrow P\overline{k}$
- $P\overline{k} \leftarrow P\overline{n}$
- $P\overline{k} \leftarrow \varepsilon$

Notar por ejemplo que la operacion de suma es '$N\overline{k} + 1$', y no '$N\overline{k} + N\overline{n}$'. Este lenguaje ni siquiera tolera ese tipo de operacion de alto nivel, al menos por ahora...

Def: **Intruccion de Control**

Las instrucciones de control aceptadas son las siguientes y **solo** las siguientes:

- IF $N\overline{k} \neq 0$ GOTO $L\overline{n}$
- IF $P\overline{k}$ BEGINS a GOTO $L\overline{n} \quad (\text{con } a \in \Sigma)$
- GOTO $L\overline{n}$
- SKIP

Def: **Instrucción de $\mathcal{S}^\Sigma$**

Una instrucción de $\mathcal{S}^\Sigma$ es una **palabra** formada por una instrucción básica (es decir una asignación o una instrucción de control), o por una palabra de la forma αI donde I es una instrucció básica y $\alpha \in \{L\overline{n} : n \in \mathbb{N}\}$

Intuitivaemnte hablando, cada instrucción puede ser una asignación o una instrucción de control, y opcionalmente puede tener al principio una label para usar control de ejecución, reminicente a lenguajes de programación con GOTO como *assembler* o *BASIC*.

Def: Ins^Σ

Definimos Ins^Σ como el conjunto de todas las instrucciones de $\mathcal{S}^\Sigma$

3.2.4 Programa

Def: **Programa**
Un programa es una palabra de la forma $I_1 I_2 \dots I_n$ donde $n \geq 1, I_1, \dots, I_n \in Ins^{\Sigma}$ y además cumple la 'ley de los GOTO'

Def: **Ley de los GOTO**
Dado un programa $P = I_1 I_2 \dots I_n$, para cada $i \in \{1, \dots, n\}$, si GOTO$L\overline{m}$ es un tramo final de I_i, entonces existe $j \in \{1, \dots, n\}$ tal que I_j empieza con la label $L\overline{m}$
Intuitivamente hablando, la ley de los GOTO es que no puede haber un salto a una label que no pertenezca al programa.

A continuación se escribe un programa de ejemplo que computa la suma de las variables numéricas $N1$ y $N2$ y 'escribe' el resultado en $N3$:

Programa: **x+y**

$$
\begin{aligned}
& N3 \leftarrow N1 && (3.1)\\
& N4 \leftarrow N2 && (3.2)\\
L1 \ & \text{IF } N4 \neq 0 \text{ GOTO } L2 && (3.3)\\
& \text{GOTO } L3 && (3.4)\\
L2 \ & N4 \leftarrow N4 \dot{-} 1 && (3.5)\\
& N3 \leftarrow N4 + 1 && (3.6)\\
& \text{GOTO } L1 && (3.7)\\
L3 \ & \text{SKIP} && (3.8)
\end{aligned}
$$

Notar que si bien en la materia escribimos el programa en distintas lineas, agregando espacios para legibilidad, estos detalles estéticos no existen en el lenguaje por lo que un programa en realidad es solo los caracteres aceptados por el lenguaje.

Por ejemplo el *verdadero* programa que hace lo anterior es:

N3←N1N4←N2L1IFN4≠0GOTOL2GOTOL3L2N4←N4$\dot{-}$1←N4+1GOTOL1L3SKIP

Def: Pro^{Σ}
Usaremos Pro^{Σ} para denotar al conjunto de todos los programas de $\mathcal{S}^{\Sigma}$

Lemma: **Unicidad de parseamiento**

Sea Γ un alfabeto finito. Se tiene que:
(a) Si $I_1 \dots I_n = J_1 \dots J_m$ con $I_1, \dots, I_n, J_1, \dots, J_m \in Ins^{\Sigma}$
Entonces
$n = m$ y $I_j = J_j \ \forall j \geq 1$
(b) Si $\mathcal{P} \in Pro^{\Sigma}$, entonces existe una única sucesión de instrucciones $I_1, \dots, I_n$ tal que $\mathcal{P} = I_1 \dots I_n$

Def: $n(\mathcal{P})$

Definimos $n(\mathcal{P})$ como la cantidad de instrucciones del programa $\mathcal{P}$

Def: $I_i^{\mathcal{P}}$

Si tenemos un programa $\mathcal{P} = I_1^{\mathcal{P}} \dots I_{n(\mathcal{P})}^{\mathcal{P}}$ tal que $I_1^{\mathcal{P}}, \dots, I_{n(\mathcal{P})}^{\mathcal{P}} \in Ins^{\Sigma}$
También definimos $I_i^{\mathcal{P}} = \varepsilon$ cuando $i = 0$ o $i > n(\mathcal{P})$

Def: **Bas**

Esta función extrae la instrucción básica de una instrucción

$$\begin{aligned} Bas : Ins^{\Sigma} &\to (\Sigma \cup \Sigma_p)^* \\ I &\to \begin{cases} J & \text{si I es de la forma L}\overline{k}\text{J con J} \in Ins^{\Sigma} \\ I & \text{caso contrario} \end{cases} \end{aligned} \tag{3.9}$$

3.3 Computación

3.3.1 Estado y Descripciones Instantáneas

Def: $\omega^{[N]}$ **y** $\Sigma^{*[N]}$

$\omega^{[\mathbb{N}]} = \{(s_1, s_2, \dots) \in \omega^N : \exists\ n \in \mathbb{N}, s_i = 0\ \forall\ i \geq n\}$
$\Sigma^{[\mathbb{N}]} = \{(\sigma_1, \sigma_2, \dots) \in \Sigma^N : \exists\ n \in \mathbb{N}, \sigma_i = \varepsilon\ \forall\ i \geq n\}$

Def: **Estado**

Un estado es un par:

$$(\vec{s}, \vec{\sigma}) = \big((s_1, s_2, \dots), (\sigma_1, \sigma_2, \dots)\big) \in \omega^{[\mathbb{N}]} \times \Sigma^{*[\mathbb{N}]}$$

Es decir que estado es un par de infinituplas que contiene la informacion de que valores tienen alojados las distintas variables.

Def: **Descripción Instantánea**

Una descripción instantánea es una terna $(i, \overrightarrow{s}, \overrightarrow{\sigma})$ tal que $(\overrightarrow{s}, \overrightarrow{\sigma})$ es un etado e $i \in \omega$

Intuitivamente $(\overrightarrow{s}, \overrightarrow{\sigma})$ describe el estado actual de las variables e i describe cual es la *siguiente* instrucción a ejecutar.

3.3.2 Computación

Def: $S_{\mathcal{P}}$

La siguiente función toma una descripción instantánea y devuelve la instrucción instantánea 'siguiente'. Dicho de otra forma es la descripción instantánea que resulta de ejecutar, si es posible, la instrucción actual del programa.

Más formalmente:

$$\begin{aligned} S_{\mathcal{P}} \ &: \omega \times \omega^{[\mathbb{N}]} \times \Sigma^{*[\mathbb{N}]} \quad \to \omega \times \omega^{[\mathbb{N}]} \times \Sigma^{*[\mathbb{N}]} \\ &i, \overrightarrow{s}, \overrightarrow{\sigma} \qquad\qquad\quad \to \text{descr inst luego de la instrucción } I_i^{\mathcal{P}} \text{ desde } (\overrightarrow{s}, \overrightarrow{\sigma}) \end{aligned} \tag{3.10}$$

También definimos la función para cuando no es posible ejecutar la instrucción de la forma: $S_{\mathcal{P}}(i, \overrightarrow{s}, \overrightarrow{\sigma}) = (i, \overrightarrow{s}, \overrightarrow{\sigma})$ si $i = 0$ o $i > n(\mathcal{P})$

Def: **Computación**

Dado un programa $\mathcal{P}$, definimos "la computación de $\mathcal{P}$ partiendo de $(\overrightarrow{s}, \overrightarrow{\sigma})$" a la infinitupla:

$$\left((1, \overrightarrow{s}, \overrightarrow{\sigma}), S_{\mathcal{P}}(1, \overrightarrow{s}, \overrightarrow{\sigma}), S_{\mathcal{P}}(S_{\mathcal{P}}(1, \overrightarrow{s}, \overrightarrow{\sigma})), \dots\right)$$

Def: **Descripción luego de t pasos**

Llamaremos "la descripción instantánea obtenida luego de t pasos partiendo de $(\overrightarrow{s}, \overrightarrow{\sigma})$" a:

$$\overbrace{S_{\mathcal{P}}(\dots S_{\mathcal{P}}}^{t \text{ veces}}(1, \overrightarrow{s}, \overrightarrow{\sigma})\dots)$$

Def: **Detención de un Programa**

Sea

$$s = \overbrace{S_{\mathcal{P}}(\dots S_{\mathcal{P}}}^{t \text{ veces}}(1, \overrightarrow{s}, \overrightarrow{\sigma})\dots)$$

Notar que s es una 3-upla del tipo $\omega \times \omega^{[\mathbb{N}]} \times \Sigma^{*[\mathbb{N}]}$

Sea $s = (i, \overrightarrow{u}, \overrightarrow{\eta}) \in \omega \times \omega^{[\mathbb{N}]} \times \Sigma^{*[\mathbb{N}]}$

Definimos que el programa $\mathcal{P}$ se detiene (luego de t pasos) partiendo desde el estado $(\overrightarrow{s}, \overrightarrow{\sigma})$ cuando $i = n(\mathcal{P}) + 1$

Intuitivamente hablando, los programas de $\mathcal{S}^{\Sigma}$ tienen **una sola** manera de detenerse. Siempre que se detienen lo hacen habiendo realizado la ultima de sus instrucciones e intentando realizar una siguiente.

El siguiente tema es **Funciones Σ-computables**, el cual este año no entró en el tercer parcial, por lo que se va a retomar en el siguiente apunte.